ERNW Newsletter 25 / March 2009

Dear Partners and Colleagues,

Welcome to the ERNW-Newsletter No. 25 covering the topic:

Malware Analysis for Business Purposes

Version 1.0 from 10th of March 2009

Author: Michael Thumann, mthumann@ernw.de

TABLE OF CONTENT

Table of Figures:

1 INTRODUCTION

Malware is still one of the most dangerous threats for companies in the year 2009, regardless if working antivirus measures are in place or not. Companies are facing automated worms and also targeted attacks against important employees that can lead to critical data breaches. New techniques to hide the malware from any antivirus program are developed and targeted attacks almost always contain customized malware to prevent them from being identified and analyzed.

Large enterprises have already started to create special response teams for analyzing these kind of malware attacks to get a better understanding what kind of information attackers are interested in and what techniques are used to cover the programs and their functionality.

This paper will introduce the different approaches how malware can be analyzed and discuss the pros and cons. It will cover online sandboxes, individually build sandbox systems with a dedicated tool set and the required protection features and also a reverse engineering approach. It will describe obfuscation techniques that are used by attackers to prevent the malware from being analyzed and possible solutions to defeat them. Finally we will give some recommendations which approach works best in a company from our point of view.

2 ONLINE SANDBOX SYSTEMS

Online sandbox systems are a good starting point for an easy and fast analysis of suspicious programs, especially in a business context. You receive your results within minutes and choosing one of the recommended vendors, it is quite accurate and can help to implement the right measures against the malware threat.

2.1 Threatexpert

This online sandbox can be reached via http://www.threatexpert.com and the service is offered for free. You can submit samples anonymously and the report will be visible for all Threatexpert users, but there is also the possibility the register a free account and keep you submission results private. It is also possible to reanalyze a malware sample that was already submitted and get your personal report.

Figure 1: Threatexpert Report

2.2 CWSandbox

The university of Mannheim runs http://www.cwsandbox.org based on the corresponding commercial product of Sunbelt Software and is also offering the service for free. Compared to Threatexpert the results are more detailed and sometimes reveal additional information that other online sandboxes don't find, but the Threatexpert report is better structured and more readable. When a sample is submitted that was already analyzed, you just get a link to the report, but the sample won't be reanalyzed. All reports are publically available and there is also no individual environment for private reports.

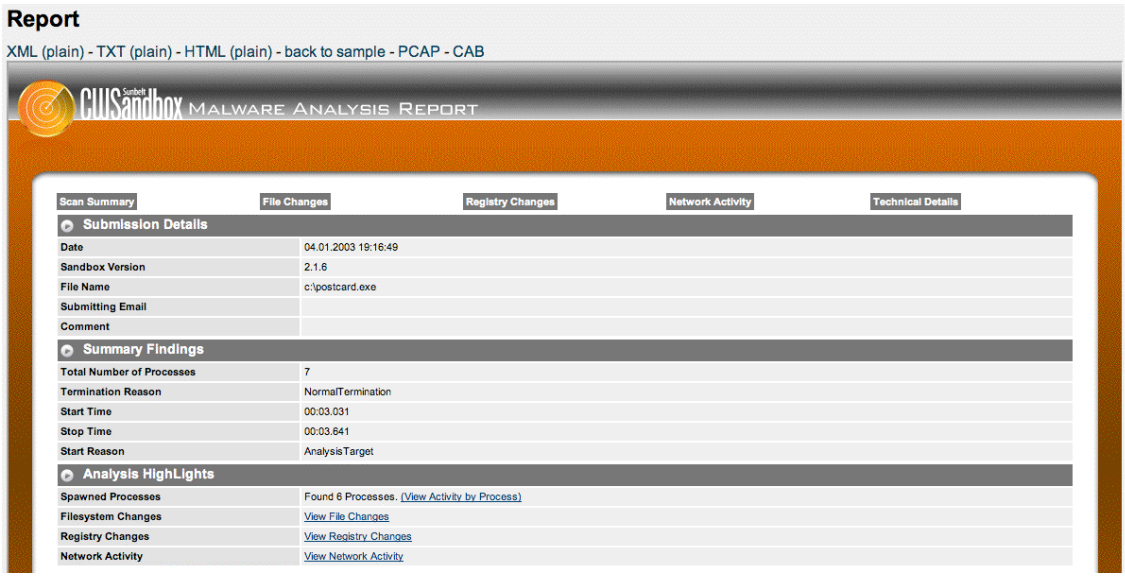

Figure 2: CWSandbox Report

2.3 Norman Sandbox

The Norman sandbox information center also offers the submission of malware samples at http://www.norman.com/microsites/nsic/Submit/en-us, but compared to the other sandbox systems you only get a marketing email back for free. The provided details about your malware are pretty poor and without any real business value, it's more or less an advertisement to buy the product.

2.4 Conclusion

The free online sandbox analysis services of Threatexpert and CWSandbox can be used for business purposes. CWSandbox provides more details, but Threatexpert has the great advantage that your results will arrive within minutes, which is quite helpful when working on an actual incident. Both services don't reveal any details about the submitter of the malware samples, but when submitting targeted attack samples the technical details might disclose information about your company like email addresses of the targeted victim.

Online sandboxes have their business value in providing fast analysis results, but also keep in mind that they have their limitations. Malware that contains anti-reverse-engineering technology like detection of virtual environments might not run in these sandbox systems or won't reveal it's damage potential due to a modified behavior.

3 INDIVIDUAL SANDBOX SYSTEM

Building your own sandbox system is the next step in a malware analyzing business process, especially if you're not allowed to submit your samples to online systems due to confidentiality requirements in your company and to prevent any kind of information disclosure. Of course there is the possibility to buy one of the recommended products of chapter 2 and run it in your own environment, but you also might have additional requirements like

- Supporting other operating systems than Windows like MAC OS X, Linux or Solaris
- Add stealth functionality to analyze malware with anti-reverse-engineering technology
- Reflect your company security controls to see the specific impact to your environment

and there might be more. But building your own sandbox is also a challenge, because some considerations have to be made to prevent any damage to your environment and control the sandbox.

3.1 Technical Requirements

Installing and running a sandbox in your own network environment requires some mandatory technical controls to prevent your sandbox from attacking other companies. On the other hand some basic functionality like file downloads or general web server access must be allowed to get proper results. An acceptable approach is to separate the possible network traffic in 2 categories:

1. Control Channel
2. Infection Channel

The Control Channel communicates e.g. with a botnet master to receive commands or download additional components to extend the functionality of the malware program, the typical communication for that is often based on HTTP or IRC. The infection channel is used to spread the malware e.g. via email, file sharing or maybe using an OS related vulnerability and corresponding protocols like SMB. Of course we're interested in both of these channels, but we have to handle the communication differently.

To prevent your sandbox from being permanently infected with the malware sample there must also be a mechanism in place to restore an initial uninfected and clean state.

So the following technical controls must be in place when setting up your own sandbox:

- Personal or network firewall to control the communication
- A dedicated DNS server to resolve DNS queries to network services under your control
- Fake servers (Mail, Windows server) to capture the infection traffic and redirect it to non-critical systems
- A restore procedure for your sandbox like Windows restore points or VMware snapshots

This recommended environment can be build using dedicated systems and network segments, if you're willing to spent some for money for it, but it's also possible to build a dedicated VMware image that contains almost all of the mentioned requirements, except the firewall functionality. The firewall should be run separated, e.g. on the system hosting the VM to ensure that it can't be circumvented or disabled by the malware. Also keep in mind that there are attack vectors against virtualized environments that might put the host system at risk, if it isn't properly secured.

3.2 The Analysis Toolset

The main motivation for running a sandbox in a business environment is to do a behavior based analysis, so we need our special tools that will fulfill the job. Again we have different categories where data has to be recorded and monitored:

- Registry access
- File system access
- Process monitoring
- API monitoring
- Network monitoring

For each category dedicated tools are available, e.g.

- Registry access: RegMon (http://technet.microsoft.com/en-us/sysinternals/bb896652.aspx)
- File system access: FileMon (http://technet.microsoft.com/en-us/sysinternals/bb896642.aspx)
- Process monitoring: ProcMon (http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx)
- API monitoring: Autodebug Professional (http://www.autodebug.com/) - Commercial Software
- Network monitoring: Wireshark (http://www.wireshark.org/)

But there are also dedicated malware analysis tools that are available for free, summarizing these categories. iDefense offers tool suites for that purpose in their download section (http://labs.idefense.com/software/malcode.php):

- SysAnalyzer: An automated malcode run time analysis tool
- Malware Analysis Pack: Toolset for rapid malware analysis (Fake DNS and Mail server)
- Multipot: An emulation based honeypot designed to capture malicious code

Installing all these tools gives you a sandbox system that can be used for doing manual analysis of the malware samples, but requires some hands-on experience with the toolset.

3.3 Conclusion

Although building your own sandbox system gives you a lot of flexibility in the analysis process and ensures the compliance to corporate security policies regarding confidentiality requirements, there are also some disadvantages:

- The analysis process is not automated
- The monitoring tools can still be detected by the malware program
- A lot of effort must be spent to ensure proper isolation of the sandbox system

From the authors point of view this approach can't be recommended in a business context where time and manpower are important factors. If there's a business need for doing malware analysis in your own environment, consider buying one of the products mentioned in chapter 2 instead.

But if you want to work with the following reverse engineering approach, building your individual sandbox is mandatory requirement.

4 REVERSE ENGINEERING

Reverse engineering comes into place when you need an in-depth analysis of your malware sample. Although in theory it might be possible to do a complete static analysis without running the malware, but when facing today's modern malware that won't work in practice. Defeating the commonly applied anti-reverse-engineering techniques requires that the malware is executed at least until a specific point to acquire a disassembly you can work with. To avoid infection of your analysis system you need a customized sandbox system as mentioned in the previous chapter. Equipped with the right toolset, a deep knowledge about the operating system and some programming skills, you're ready to start the reversing process following a structured approach to accomplish your task in a reasonable amount of time.

4.1 Required Tools for Reversing Malware

Even if there are tons of helpful tools out there in the Internet, that can make the reverse engineers life much easier, the following toolset is a minimum requirement for reversing malware:

- IDA Pro 5.4: Commercial Disassembler available at http://www.hex-rays.com/
- Hex-Rays: Commercial Decompiler Plugin for IDA Pro available at http://www.hex-rays.com/
- X86emu: x86 Emulator Plugin for IDA Pro available at https://sourceforge.net/projects/ida-x86emu/
- RDG Packer Detector: Program for detecting code obfuscators available at http://rdgsoft.8k.com/IndexIngles.html
- Bochs 2.3.7: Virtualizing Software and PC Emulator available at http://bochs.sourceforge.net/
- OllyDBG: Windows Ring 3 Debugger available at http://www.ollydbg.de/
- Ollydump: OllyDBG plugin that dumps a program from memory available at http://www.woodmann.com/collaborative/tools/index.php/OllyDump
- Phant0m: OllyDBG plugin for hiding the debugger available at http://www.woodmann.com/collaborative/tools/index.php/PhantOm

4.2 The structured approach

When doing reverse engineering for business purposes, you always have a limited amount of time to accomplish your tasks, so a structured approach is needed to stay focused on the job. So here are the steps that are needed to reverse malware, some of them are explained later on in more detail.

1. Get hands on your malware sample
2. Prepare your sandbox
3. Detect code obfuscation
4. Defeat code obfuscation
5. Detect anti-reversing tricks
6. Defeat anti-reversing tricks
7. Analyze what the malware is doing

4.3 Detect code obfuscation

More than 80% of the malware samples are obfuscated in some way, so one mandatory step before starting the reversing process is to detect EXE packers and encryptors that must be defeated. There are two possibilities, using a packer detector or take a direct look into the disassembly and spot the typical signs for code obfuscation. Of course the packer detector is the easier approach, so let's have a look at RDG Packer Detector first. The tool is scanning for well-

known signatures of commonly used packers and encryptors. Here is an example scanning the conficker worm:

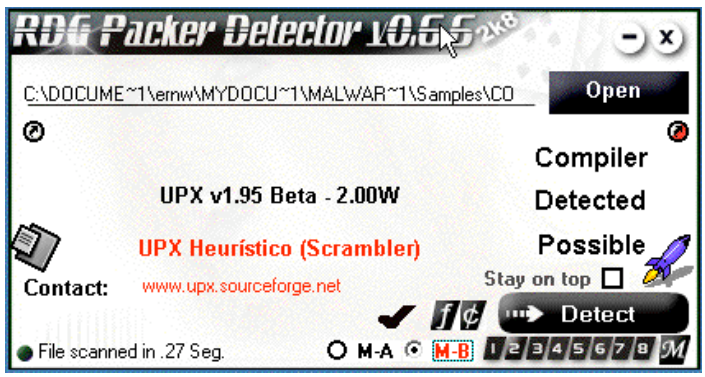

Figure 3: RDG scanning conficker

The second possibility is to load the malware into IDA Pro and examine the disassembly for typical signs of packers:

```
UPX1:10019B40 ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
UPX1:10019B40                 public DllEntryPoint
UPX1:10019B40 DllEntryPoint:
UPX1:10019B40                 cmp     byte ptr [esp+8], 1
UPX1:10019B45                 jnz     loc_10019D0D
UPX1:10019B4B                 pusha
UPX1:10019B4C                 mov     esi, offset dword_10006000
UPX1:10019B51                 lea     edi, [esi-5000h]
UPX1:10019B57                 push    edi
UPX1:10019B58                 jmp     short loc_10019B6A
UPX1:10019B58 ; ---------------------------------------------------------------------------
UPX1:10019B5A                 align 10h
```

Figure 4: Initial Disassembly with IDA

IDA recognizes UPX and some other packers out of the box, but most of the packers are harder to identify. The next example shows a part of the code segment of the sasser worm after the initial IDA disassembly:

```
.text:00401000 _text          segment para public 'CODE' use32
.text:00401000                assume cs:_text
.text:00401000                ;org 401000h
.text:00401000                assume es:nothing, ss:nothing, ds:_text, fs:nothing, gs:nothing
.text:00401000                dd 6F4000A1h, 0C06900040h, 30043FDh, 9EC30500h, 0B4A30027h
.text:00401000                dd 0E8C1000Fh, 7FFF2510h, 0C3050000h, 424448Bh, 81C311B0h
.text:00401000                dd 1902CECh, 0F8D5810h, 68501700h, 0FF050901h, 8550F015h
.text:00401000                dd 0A0C48116h, 8B55C315h, 5756EC02h, 0C30875FFh, 80512014h
.text:00401000                dd 89C8BE0Fh, 5085845h, 30A5855h, 0FC0B7075h, 0B8C91785h
.text:00401000                dd 7D012600h, 85C80302h, 0F80570FFh, 0D0F0D285h, 0F0F0F685h
.text:00401000                dd 8302016Ah, 75587FF9h, 7519C20Ch, 7517C041h, 2CF03B3Dh
.text:00401000                dd 2113774h, 8132740Ah, 33C0ACF9h, 830A7500h, 7E0FFF0Bh
.text:00401000                dd 20050427h, 118E207Ch, 810881C0h, 771A8FFh, 0F631074h
.text:00401000                dd 8F0AA3A9h, 27500FEh, 5E5FC032h, 0E38CB45Dh, 53CB00ACh
.text:00401000                dd 858D0B93h, 680038A8h, 5000FF38h, 0E0189CB1h, 4275C085h
.text:00401000                dd 10F315BCh, 0F88BC024h, 3BF633h, 8B2D74FEh, 30390C47h
.text:00401000                dd 8B267401h, 30FF0604h, 0A01C1963h, 58E853D8h, 5403807h
.text:00401000                dd 7559C084h, 831EC015h, 3C8304C6h, 0DC670658h, 8B03EB00h
.text:00401000                dd 0DB85085Dh, 53030174h, 0F86805EBh, 0FA296769h, 9C501863h
.text:00401000                dd 0C95B2D7Fh, 0EC830580h, 6A7D8110h, 67458D10h, 443C78DEh
.text:00401000                dd 834B0212h, 0C7660CC4h, 20EC0h, 3D9BBD68h, 36510816h
```

Figure 5: Sasser Disassembly

IDA couldn't disassemble this part of the code segment during the initial analysis and has marked it as data, which is a reliable sign for an EXE packer or encryptor.

4.4 Defeat code obfuscation

To go on with the reversing task the code obfuscation must be defeated, that means the malware must be unpacked or decrypted. As long as UPX is used this is quite easy, because you can use UPX for unpacking too:

```
hal9000:conficker mthumann$ upx -d conficker.dll -o conficker-unpacked.dll
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2008
UPX 3.03        Markus Oberhumer, Laszlo Molnar & John Reiser   Apr 27th 2008

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    169591 <-    162423   95.77%    win32/pe     conficker-unpacked.dll

Unpacked 1 file.
```

But life isn't always that easy and unpacking the malware is one of the greater challenges when reversing malware. Very often the malware must be run within a debugger to let it unpack or decrypt itself and then take a memory dump of the program. OllyDBG is a good debugger for doing that and the memory dump can be done with the plugin OllyDump. Of course the malware should only be run in your prepared sandbox.

Looking at the conficker disassembly after unpacking reveals that the code is still obfuscated. We can spot a lot of offsets for function calls in the data segment

```
.data:10006284 off_10006284    dd offset unk_10016878  ; DATA XREF: sub_1000204A+1A01↑r
.data:10006288 off_10006288    dd offset sub_1000429C  ; DATA XREF: sub_100027E5-166C↑r
.data:1000628C off_1000628C    dd offset sub_10001A83  ; DATA XREF: sub_10002DCD-903↑r
.data:10006290 off_10006290    dd offset sub_10001608  ; DATA XREF: sub_100034BE+A↑r
.data:10006294 off_10006294    dd offset sub_1000417F  ; DATA XREF: sub_10002FA6+1B↑r
.data:10006298 off_10006298    dd offset sub_100044F4  ; DATA XREF: sub_10002032-2A4↑r
```

Figure 6: Offsets for Functions

and also bytes that look like encrypted data at the first glance:

```
.data:100063B0  75 0B BB 13 97 19 A9 37  BD B2 82 4E C0 BE 02 82  u■»■■↓©7½²■N└¾■■
.data:100063C0  C8 26 2F 70 F0 75 33 49  CF 69 EC F1 80 6B 23 AD  È&/pðu3IÏiìñ■k#-
.data:100063D0  64 69 15 77 CD 27 B1 E9  FD 75 3E B9 21 61 AD 01  di§wÍ'±éýu>¹!a-■
.data:100063E0  61 47 68 4C D4 68 B7 E9  F3 0A 46 DD 25 08 C3 21  aGhLÔh·éó■FÝ%■Ã!
```

Figure 7: Encrypted data?

We can identify typical functions like VirtualAlloc and VirtualProtect that are used to write decrypted code directly to the memory and then call it:

```
.text:10003972   push   edx              ; dwSize
.text:10003973   push   ecx              ; lpAddress
.text:10003974   call   ds:VirtualAlloc
.text:1000397A   test   eax, eax
.text:1000397C   jz     loc_10002A61
.text:10003982   jmp    loc_10003258

.text:100038BE   push   eax              ; dwSize
.text:100038BF   push   edx              ; lpAddress
.text:100038C0   call   ds:VirtualProtect
.text:100038C6   mov    edi, ds:rand
.text:100038CC   call   edi ; rand
.text:100038CE   mov    ecx, dword 100167C4
```

Figure 8: Suspicious Functions

So it looks like we still have some work to do to deobfuscate the code. Because doing that with a static approach, just looking at the assembler code, will take too much time, we start OllyDBG, load the binary and set breakpoints at the VirtualAlloc and VirtualProtect function calls. When the breakpoints are reached we can find the deobfuscated code in the allocated memory region. For getting a proper disassembly we have to rebuild the PE header and the import table for the code and then we are ready to start the analysis.

A very detailed walkthrough of the conficker unpacking steps can be found at http://earlmarcus.blogspot.com/2009/01/unpacking-confickerdownadup.html.

4.5 Detect anti-reversing tricks

The creators of malware are aware of the methodologies used for unpacking their malware and are protecting it with anti-reversing tricks to make the analysis process much harder. Often used tricks are:

- Detecting Debuggers using the Windows API call *IsDebuggerPresent*
- Detecting Virtualization e.g. looking for specific hardware or registry keys
- Detecting Instrumentation e.g. with FindWindow("FilemonClass", NULL)
- Dynamically Computed Target Addresses are used to ensure that the execution flow can only be followed at runtime
- Targeted Attacks against the Analysis Tools e.g. vulnerabilities in IDA and OllyDBG

And there are more tricks and also variants of the mentioned ones. Here is code snippet of conficker checking for the presence of a debugger using an internal Windows API call (which by the ways is also a sign for the usage of advanced code obfuscation techniques):

```
.text:10002F7E
.text:10002F7E loc_10002F7E:                          ; CODE XREF: sub_1000296A+24↑j
.text:10002F7E                call    ds:IsDebuggerPresent
.text:10002F84                mov     dword_100169F4, eax
```

Figure 9: Example Debugger Check

These anti-reversing tricks can be detected by examining the disassembly for suspicious API calls and observing the runtime behavior of the malware in the debugger e.g. malware crashes when a debugger is attached.

4.6 Defeat anti-reversing tricks

Defeating these anti-reversing tricks is the hardest challenge when analyzing malware manually, especially when more of these tricks are combined. Nevertheless there are some approaches to get a reliable result, because finally the malware has to be executed and must be stored in memory in an executable form which means in readable machine code. These approaches require to run the malware and to prevent a malware outbreak, this has to be done in a controlled environment, and so for reversing malware an individual sandbox system is a must.

Very often this is done using OllyDBG. OllyDBG has a plugin interface, so it can be extended with useful plugins for hiding the debugger and defeat debugger detection tricks. When working with OllyDBG we mainly use Phant0m for that purpose, it is quite well maintained and new functionality is added frequently. Here's an actual screenshot which techniques are supported by phant0m at the time of writing:

Figure 10: Phant0m Anti-Anti-Debugging

Defeating anti-vm tricks is much harder because many techniques are used to detect a virtualized environment. Hiding your virtual machine contains approaches like

- Don't install VM Tools
- Change the MAC Address of your NIC
- Usage of virtualization software that isn't very common
- Binary patch the malware to NOP the vm detection routines

After managing to defeat all these nasty anti-reversing tricks you are ready to dump the binary from memory to disk.

4.7 Analyze what the malware is doing

After acquirering your deobfuscated disassembly the reverse engineering process can be started. Typically malware contains two parts of functionality:

- Infection and spreading
- Damage

Both of them are under the scope of the reverse engineer, because in a business context you want to protect you infrastructure from getting infected and you want to figure out what kind of damage is done to the systems.

First we will have an exemplary look at the infection and spreading part. Obviously conficker loops to a pool of IP addresses:

```
.text:003A9BE9                  push    offset aD_D_D_D_0 ; "%d.%d.%d.%d"
.text:003A9BEE                  lea     eax, [ebp+6Ch+var_88]
.text:003A9BF1                  push    80h ; '€'
.text:003A9BF6                  push    eax
.text:003A9BF7                  call    esi ; _snprintf
.text:003A9BF9                  push    ebx
.text:003A9BFA                  mov     [ebp+6Ch+var_9], 0
.text:003A9BFE                  call    sub_3A8DB4
```

Figure 11: Passing IP to function

It constructs a connection string to the remote IPC$ share of the IP address

```
.text:003A9CEA                  push    offset aSIpc    ; "\\\\%s\\IPC$"
.text:003A9CEF                  lea     eax, [ebp+6Ch+var_188]
.text:003A9CF5                  push    100h
.text:003A9CFA                  push    eax
.text:003A9CFB                  call    esi ; _snprintf
```

Figure 12: Remote Share being constructed

and tries to get the remote OS version:

```
.text:003A9C69                  mov     [ebp+6Ch+VersionInformation.dwOSVersionInfoSize], 9Ch ; '▮'
.text:003A9C73                  lea     edi, [ebp+6Ch+VersionInformation.dwMajorVersion]
.text:003A9C79                  rep stosd
.text:003A9C7B                  lea     eax, [ebp+6Ch+VersionInformation]
.text:003A9C81                  push    eax             ; lpVersionInformation
.text:003A9C82                  call    ds:GetVersionExA
```

Figure 13: Remote GetVersion Call

If the remote system is vulnerable to the MS08-067 Vulnerability, conficker exploits it automatically and infects the system. This is just a small snapshot of the infection part of conficker, let's move on to the damage part.

After deleting a service, a new one is created and started to ensure that the malware will survive a reboot of the infected system:

```
.text:003A7F65                  push    offset Password ; lpPassword
.text:003A7F6A                  push    esi             ; lpServiceStartName
.text:003A7F6B                  push    esi             ; lpDependencies
.text:003A7F6C                  push    esi             ; lpdwTagId
.text:003A7F6D                  push    esi             ; lpLoadOrderGroup
.text:003A7F6E                  push    [ebp+lpBinaryPathName] ; lpBinaryPathName
.text:003A7F71                  push    esi             ; dwErrorControl
.text:003A7F72                  push    3               ; dwStartType
.text:003A7F74                  push    1               ; dwServiceType
.text:003A7F76                  push    0F01FFh         ; dwDesiredAccess
.text:003A7F7B                  push    [ebp+lpDisplayName] ; lpDisplayName
.text:003A7F7E                  push    [ebp+lpDisplayName] ; lpServiceName
.text:003A7F81                  push    eax             ; hSCManager
.text:003A7F82                  call    ds:CreateServiceA
.text:003A7F88                  mov     edi, ds:3A1068h
.text:003A7F8E                  mov     ebx, eax
.text:003A7F90                  cmp     ebx, esi
.text:003A7F92                  jz      short loc_3A7FA2
.text:003A7F94                  push    esi             ; lpServiceArgVectors
.text:003A7F95                  push    esi             ; dwNumServiceArgs
.text:003A7F96                  push    ebx             ; hService
.text:003A7F97                  call    ds:StartServiceA
```

Figure 14: Installing Backdoor

Of course conficker also creates a registry key in one of the autostart sections:

```
.text:003AD8B9                  push    offset aSoftwareMicr_0 ; Software\Microsoft\Windows\CurrentVersion\Run
.text:003AD8BE                  push    esi             ; hKey
.text:003AD8BF                  call    ds:RegCreateKeyExA
```

Figure 15: Create Autostart RegKey

In the next step subkeys are generated that contain all required information to start parts of the malware each time when Windows starts.

```
.text:003AD6AC                  call    ds:RegCreateKeyExW
.text:003AD6B2                  test    eax, eax
.text:003AD6B4                  jnz     short loc_3AD6EA
.text:003AD6B6                  push    [ebp+arg_0]
.text:003AD6B9                  call    ebx
.text:003AD6BB                  pop     ecx
.text:003AD6BC                  lea     eax, [eax+eax+2]
.text:003AD6C0                  push    eax             ; cbData
.text:003AD6C1                  push    [ebp+arg_0]     ; lpData
.text:003AD6C4                  push    2               ; dwType
.text:003AD6C6                  push    edi             ; Reserved
.text:003AD6C7                  push    offset aServicedll ; "ServiceDll"
.text:003AD6CC                  push    [ebp+phkResult] ; hKey
.text:003AD6CF                  call    esi ; RegSetValueExW
.text:003AD6D1                  push    [ebp+phkResult] ; hKey
.text:003AD6D4                  call    ds:RegCloseKey
```

Figure 16: Add registry subkeys

Conficker also compromises installed antivirus programs and the firewall settings to keep the malware stealthy. Here's a code snippet disabling important security services:

```
.text:003A78BE loc_3A78BE:                              ; CODE XREF: mtCheckAntiMalware1+129↑j
.text:003A78BE                  push    offset aWscsvc  ; "wscsvc"
.text:003A78C3                  call    mtStopAndDisableService
.text:003A78C8                  mov     [esp+1ACh+var_1AC], offset aSoftwareMicros ; "Software\\Microsoft\\Windows\\
.text:003A78CF                  mov     esi, 80000002h
.text:003A78D4                  push    esi
.text:003A78D5                  call    dword ptr ds:3A1330h
.text:003A78DB                  push    offset aWuauserv ; "wuauserv"
.text:003A78E0                  call    mtStopAndDisableService
.text:003A78E5                  mov     [esp+1ACh+var_1AC], offset aBits ; "BITS"
.text:003A78EC                  call    mtStopAndDisableService
.text:003A78F1                  mov     [esp+1ACh+var_1AC], offset aWindefend ; "WinDefend"
.text:003A78F8                  call    mtStopAndDisableService
.text:003A78FD                  mov     [esp+1ACh+var_1AC], offset aWindowsDefende ; "Windows Defender"
.text:003A7904                  push    offset aSoftwareMicr_0 ; "Software\\Microsoft\\Windows\\CurrentVersi"...
.text:003A7909                  push    esi
.text:003A790A                  call    dword ptr ds:3A1334h
.text:003A7910                  push    offset aErsvc   ; "ERSvc"
.text:003A7915                  call    mtStopAndDisableService
.text:003A791A                  mov     [esp+1ACh+var_1AC], offset aWersvc ; "WerSvc"
.text:003A7921                  call    mtStopAndDisableService
```

Figure 17: Disable AV services

While analyzing the malware we also stumbled about an exploitable buffer overflow when conficker is requesting the external IP address of the infected system from an online service, so it looks like even malware isn't developed in a secure manner:

```
.text:003B51F3                  push    [esp+8+arg_4]
.text:003B51F7                  lea     eax, [esi+8]
.text:003B51FA                  push    eax
.text:003B51FB                  call    j_strcpy
```

Figure 18: Buffer Overflow in conficker

This was just a short journey into the analyzing part of the reverse engineering approach. If you're interested in more details about conficker, you can find a very good analysis at http://mtc.sri.com/Conficker/.

4.8 Conclusion

While reversing your malware sample gives you the most detailed information, it also requires the most amount of time for analysis and very skilled people to do this job. So in a business context the approach isn't used too often, mainly just for targeted attacks against VIPs in the organization and accomplished by external partners that offer this kind of service.

5 RECOMMENDATION

Doing malware analysis especially in a business context is getting more and more common, but it must be accomplished in a reasonable amount of time to minimize the impact of a malware outbreak and to take care of economic requirements. All of the mentioned methods have their pros and cons, some of them are easy to use and others require an in-depth knowledge of reverse engineering. So, which is the right one to choose? A good solution would be to implement the malware analysis in your incident response process and define different actions for the different types of malware. Of course different types must be put into categories, e.g.

1. Known malware (detected by antivirus solutions), targeting all computer users
2. Unknown malware (not yet detected by antivirus solutions), targeting all computer users
3. Known (already analyzed) targeted malware, targeting your organization
4. Unknown (not yet analyzed) targeted malware, targeting your organization
5. Known (already analyzed) targeted malware, targeting VIPs in your organization
6. Unknown (not yet analyzed) targeted malware, targeting VIPs in your organization

The next step is to define a malware analysis action plan for these categories:

Category	Action	Tool
1.	Nothing, should be detected by AV solution	Antivirus
2.	Acquire sample and analyze	Online sandbox
3.	Inform all users and ensure that AV is up to date	Antivirus
4.	Acquire sample and analyze. Create custom signature for AV and deploy.	Online sandbox (depending on your internal policies) or internal sandbox / Antivirus
5.	Inform targeted users and ensure that AV is up to date	Antivirus
6.	Acquire sample and analyze. Create custom signature for AV and deploy.	Internal sandbox / RE of malware (maybe using partners) / Antivirus

This is just a generic example, how all this stuff can be implemented as business process and how to define mandatory steps. There might be additional requirements depending on the main purpose of your organization and the processed data.

6 SUMMARY

Malware analysis isn't a typical field of antivirus companies anymore. Targeted attacks against organizations require them to deal with this threat and implement corresponding procedures, a usable tool set reflecting the individual knowledge and user awareness. Organizations should be prepared before the first major impact and address this topic in their IT-Security policies and procedures.

Kind regards,
[Michael Thumann].

ERNW GmbH
Michael Thumann
Senior Security Consultant

ERNW Enno Rey Netzwerke GmbH
Breslauer Str. 28
69124 Heidelberg
Tel. +49 6221 480390
Fax +49 6221 419008
www.ernw.de